Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 196 (2008) 3–18
www.elsevier.com/locate/entcs

Formalising in Nominal Isabelle Crary's Completeness Proof for Equivalence Checking

Julien Narboux[1] and Christian Urban[2]

TU Munich, Germany

Abstract

In the book on Advanced Topics in Types and Programming Languages, Crary illustrates the reasoning technique of logical relations in a case study about equivalence checking. He presents a type-driven equivalence checking algorithm and verifies its completeness with respect to a definitional characterisation of equivalence. We present in this paper a formalisation of Crary's proof using Isabelle/HOL and the nominal datatype package.

Keywords: logical relations, proof assistants, formalisations, Isabelle/HOL, nominal logic work.

1 Introduction

Logical relations are a powerful reasoning technique for establishing properties about programming languages. The idea of logical relations goes back to Tait [8] and is usually employed for showing strong normalisation results. However this technique has wide applicability. Crary illustrates this by using a logical relation argument to prove completeness of an equivalence checking algorithm [3]. One reason for formalising proofs involving logical relations is that they are fairly intricate: First they require a logic that is sufficiently strong (see comment in [4, Page 58]). Also in the final step of such proofs, one has to establish by induction a property under a closing substitution. These substitutions might, however, interfere with binders and one has to be careful that the proof covers all cases that are required by the induction. We will show in this formalisation that there are a few places where one

[1] Email: narboux(at)in.tum.de
[2] Email: urbanc(at)in.tum.de

doi:10.1016/j.entcs.2007.09.014

has to pay attention to this issue and that the strong induction principles [10] that have the variable convention already built in are quite convenient to get the formal arguments through.

There have already been a number of formalisations of proofs involving logical relations. For example Altenkirch [1] formalises the usual strong normalisation proof for System F in the theorem prover LEGO. To our knowledge all these formalisations use de Bruijn indices to represent α-equated terms. We attribute this to the fact that proofs using logical relations heavily rely on terms being a representation for α-equivalence classes. We assume that this is the reason why a formalisation based on a concrete (un-quotioned) representation has never been attempted.

One practical reason why we do not wish to formalise Crary's proof using de Bruijn indices is that we like to stay as faithful as possible to the source and thus do not need to invent any of the formal arguments ourselves. This intention materialised quite a bit in our formalisation, except in one place where we developed a completely different argument than the one Crary had mind, but did not completely spell out its details (we found this out after we completed the formalisation by communicating with Crary about our proof). Even so we also had to spend considerable work to implement the informal rules presented by Crary and to justify that our implementation captures the intended behaviour of these rules.

Our formal proof is carried out in Isabelle/HOL and relies much on the infrastructure provided by the nominal datatype package [9,10,11]. This package uses many ideas from the nominal logic work by Pitts [6]. The ability to directly define in the nominal datatype package α-equivalent terms and obtain automatically recursion combinators and strong induction principles that have the usual variable convention already built was of great help in our formalisation. There is one place were we had to derive manually some infrastructure, which we hope can be derived automatically in the future. In the rest of the paper we give a guided tour through our formalisation.

2 Terms, Types and Substitution

Terms, types and substitutions are relatively standard and follow closely Crary's notes. Terms are given by the grammar

Definition 2.1 (Terms)

$trm ::= Var\ name \mid App\ trm\ trm \mid Lam\ name.trm \mid Const\ nat \mid Unit$

where in the *Lam*-clause, as usual, a variable is bound; there is also an infinite supply of constants all represented by natural numbers. By stating this definition in the nominal datatype package we immediately obtain α-equivalent terms. Types are given by the grammar

Definition 2.2 (Types) $ty ::= TBase \mid TUnit \mid ty \rightarrow ty$

where there is no binding. We define the usual size function for types (details omitted), as this will be the measure over which we define the logical relation later

on.

The most important operation we need for our terms is that of applying simultaneous substitutions, which we represent as finite lists of (*name*,*trm*)-pairs. Crary defines them as functions from some set of variables to terms. One reason for our choice is that it is easier to deal with finitary structures in the nominal datatype package than with infinite ones (functions are considered as infinitary structures and would require additional theorem prover code). Using our list representation we define:

Definition 2.3 (Simultaneous Substitution)

$$\begin{array}{lcll} \theta(Var\ x) & = & lookup\ \theta\ x & \\ \theta(App\ t_1\ t_2) & = & App\ \theta(t_1)\ \theta(t_2) & \\ \theta(Lam\ x.t) & = & Lam\ x.\theta(t) & \text{provided } x\ \#\ \theta \\ \theta(Const\ n) & = & Const\ n & \\ \theta(Unit) & = & Unit & \end{array}$$

where in the first clause we use the auxiliary function *lookup* defined by the clauses:

$$\begin{array}{lcl} lookup\ []\ x & = & Var\ x \\ lookup\ ((y,\ T)::\theta)\ x & = & if\ x = y\ then\ T\ else\ lookup\ \theta\ x \end{array}$$

Single substitutions are a derived concept by defining $e[x{:=}e'] \stackrel{\text{def}}{=} [(x,\ e')](e)$ with $[(x,\ e')]$ being a singleton list.

Note that in the *Lam*-clause we attach the side-condition about x being fresh for θ (written $x\ \#\ \theta$), which is equivalent to x being not free in the list of (*name*,*trm*)-pairs. Despite imposing this side-condition, the definition above yields a total function, since we work with α-equivalence classes where renamings are always possible. Because we define a function over the structure of α-equated terms, we must be careful to not introduce any inconsistencies [9]. The reason is that we can specify functions over the structure of such terms that do not respect α-equivalence (for example the function that calculates the bound names of a term or returns the immediate subterms) and consequently lead to inconsistencies in Isabelle/HOL. In our formalisation this means that we have to give two four-line proofs that ensure that simultaneous substitutions respect α-equivalence.

3 Typing and Definitional Equivalence

Next, we define the typing judgements for our terms. In order to stay faithful to Crary's notes we introduce the notion for when a typing context Γ is *valid*, namely when it includes only a single association for every variable occurring in Γ. Again we use lists to represent these typing contexts; this time because Isabelle/HOL does not provide out-of-the-box a type of finite sets. Using the lists we can define the notion of validity by the two rules:

$$\frac{}{valid\ []} \qquad \frac{valid\ \Gamma \qquad x\ \#\ \Gamma}{valid\ ((x,\ T)::\Gamma)}$$

where we attach in the second rule the side-condition that x must be fresh for Γ,

which in case of our typing contexts is equivalent to x not occurring in Γ. The typing rules are then defined as:

$$\frac{valid\ \Gamma \qquad (x,\ T) \in \Gamma}{\Gamma \vdash Var\ x : T}\text{T-Var} \qquad \frac{\Gamma \vdash e_1 : T_1 \to T_2 \qquad \Gamma \vdash e_2 : T_1}{\Gamma \vdash App\ e_1\ e_2 : T_2}\text{T-App}$$

$$\frac{x\ \#\ \Gamma \qquad (x,\ T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash Lam\ x.t : T_1 \to T_2}\text{T-Lam}$$

$$\frac{valid\ \Gamma}{\Gamma \vdash Const\ n : TBase}\text{T-Const} \qquad \frac{valid\ \Gamma}{\Gamma \vdash Unit : TUnit}\text{T-Unit}$$

where we ensure that only valid contexts appear in typing judgements by including validity in the rules for variables and Units. To preserve validity in the rule T-Lam, we have the side-condition that x must be fresh for Γ. (We can infer this freshness condition also from the premise $(x,\ T_1)::\Gamma \vdash t : T_2$ and the fact that in typing-judgements contexts are always valid, but this requires a side-lemma.) In rule T-Var we use the notation $(x,\ T) \in \Gamma$ to stand for list-membership.

The completeness of the typing algorithm is proved with respect to some rules characterising definitionally the equivalence between typed terms. The corresponding judgements Crary is using for this are of the form $\Gamma \vdash s \equiv t : T$ where s and t are terms and T is a type. We formalise his rules of definitional equivalence as follows:

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t \equiv t : T}\text{Q-Refl} \qquad \frac{\Gamma \vdash t \equiv s : T}{\Gamma \vdash s \equiv t : T}\text{Q-Symm}$$

$$\frac{\Gamma \vdash s \equiv t : T \qquad \Gamma \vdash t \equiv u : T}{\Gamma \vdash s \equiv u : T}\text{Q-Trans}$$

$$\frac{\Gamma \vdash s_1 \equiv t_1 : T_1 \to T_2 \qquad \Gamma \vdash s_2 \equiv t_2 : T_1}{\Gamma \vdash App\ s_1\ s_2 \equiv App\ t_1\ t_2 : T_2}\text{Q-App}$$

$$\frac{x\ \#\ \Gamma \qquad (x,\ T_1)::\Gamma \vdash s_2 \equiv t_2 : T_2}{\Gamma \vdash Lam\ x.s_2 \equiv Lam\ x.t_2 : T_1 \to T_2}\text{Q-Abs}$$

$$\frac{x\ \#\ (\Gamma,\ s_2,\ t_2) \qquad (x,\ T_1)::\Gamma \vdash s_1 \equiv t_1 : T_2 \qquad \Gamma \vdash s_2 \equiv t_2 : T_1}{\Gamma \vdash App\ (Lam\ x.s_1)\ s_2 \equiv t_1[x{:=}t_2] : T_2}\text{Q-Beta}$$

$$\frac{x\ \#\ (\Gamma,\ s,\ t) \qquad (x,\ T_1)::\Gamma \vdash App\ s\ (Var\ x) \equiv App\ t\ (Var\ x) : T_2}{\Gamma \vdash s \equiv t : T_1 \to T_2}\text{Q-Ext}$$

$$\frac{\Gamma \vdash s : TUnit \qquad \Gamma \vdash t : TUnit}{\Gamma \vdash s \equiv t : TUnit}\text{Q-Unit}$$

Validity of the typing contexts are implied by the validity in the typing rules, which are included in the premises of Q-Refl and Q-Unit, and by having the side-condition about x being fresh for Γ in Q-Abs, Q-Beta and Q-Ext.

Comparing our rules with the ones given by Crary, slightly unusual are the side-conditions $x\ \#\ (s_2,\ t_2)$ in the rule Q-Beta and $x\ \#\ (s,\ t)$ in the rule Q-Ext. In the former case we can relatively easily show that our Q-Beta is equivalent to

$$\frac{(x,\ T_1)::\Gamma \vdash s_1 \equiv t_1 : T_2 \qquad \Gamma \vdash s_2 \equiv t_2 : T_1}{\Gamma \vdash App\ (Lam\ x.s_1)\ s_2 \equiv t_1[x::=t_2] : T_2}\text{Q-Beta'}.$$

However this requires explicit α-conversions and the fact that all typing contexts in the definitional equivalence judgements are valid. In light of this equivalence, the question arises why we insist on the more restricted rule: The reason is that based on those constraints the nominal datatype package can automatically derive a strong induction principle that has the variable convention already built in. This will be very convenient in some proofs later on. To do the same without those constraints is possible, but slightly more laborious.

In case of Q-Ext the side-conditions represent the fact that the extensionality rule should hold for a fresh variable x only. By imposing $x\ \#\ (\Gamma,\ s,\ t)$ we can show that Q-Ext is equivalent to

$$\frac{\forall x.\ x\ \#\ \Gamma \longrightarrow (x,\ T_1)::\Gamma \vdash App\ s\ (Var\ x) \equiv App\ t\ (Var\ x) : T_2}{\Gamma \vdash s \equiv t : T_1 \rightarrow T_2}\text{Q-Ext'}$$

The argument for this uses the some/any-property from [6] and relies on the fact that the definitional equivalence is equivariant; by this we mean it is invariant under swapping of variables, namely $\Gamma \vdash s \equiv t : T$ implies $(x\ y)\cdot\Gamma \vdash (x\ y)\cdot s \equiv (x\ y)\cdot t : T$ for all x and y (see [10,11] for more details). The side-conditions in Q-Ext are not explicitly given by Crary and the equivalence with Q-Ext' gave us confidence to have captured with them the "idea" of an extensionality rule.

4 The Equivalence Checking Algorithm

One feature of Crary's equivalence checking algorithm is that it includes a fair amount of optimisations, in the sense that in some circumstances two lambda terms are not completely normalised but only transformed into a weak-head normal-form. For this Crary introduces the following four rules:

$$\frac{}{App\ (Lam\ x.t_1)\ t_2 \rightsquigarrow t_1[x::=t_2]}\text{QAR-Beta} \qquad \frac{t_1 \rightsquigarrow t_1'}{App\ t_1\ t_2 \rightsquigarrow App\ t_1'\ t_2}\text{QAR-App}$$

$$\frac{s \rightsquigarrow t \qquad t \Downarrow u}{s \Downarrow u}\text{QAN-Reduce} \qquad \frac{t \not\rightsquigarrow}{t \Downarrow t}\text{QAN-Normal}$$

The algorithm is then defined by two mutual recursive judgements, called respectively *algorithmic term equivalence* and *algorithmic path equivalence*. The former is written as $\Gamma \vdash s \Leftrightarrow t : T$ and the latter as $\Gamma \vdash s \leftrightarrow t : T$. Their rules are

$$\frac{s \Downarrow p \qquad t \Downarrow q \qquad \Gamma \vdash p \leftrightarrow q : TBase}{\Gamma \vdash s \Leftrightarrow t : TBase}\text{QAT-Base}$$

$$\frac{x\ \#\ (\Gamma,\ s,\ t) \qquad (x,\ T_1)::\Gamma \vdash App\ s\ (Var\ x) \Leftrightarrow App\ t\ (Var\ x) : T_2}{\Gamma \vdash s \Leftrightarrow t : T_1 \rightarrow T_2}\text{QAT-Arrow}$$

$$\frac{valid\ \Gamma}{\Gamma \vdash s \Leftrightarrow t : TUnit}\text{QAT-One}$$

$$\frac{valid\ \Gamma \qquad (x,\ T) \in \Gamma}{\Gamma \vdash Var\ x \leftrightarrow Var\ x : T}\text{QAP-Var}$$

$$\frac{\Gamma \vdash p \leftrightarrow q : T_1 \rightarrow T_2 \qquad \Gamma \vdash s \Leftrightarrow t : T_1}{\Gamma \vdash App\ p\ s \leftrightarrow App\ q\ t : T_2}\text{QAP-App}$$

$$\frac{valid\ \Gamma}{\Gamma \vdash Const\ n \leftrightarrow Const\ n : TBase}\text{QAP-Const}$$

following quite closely Crary's definition. One difference, however, is the inclusion of the validity predicate in the rules QAT-One, QAP-Var and QAP-Const ensuring that only valid typing contexts appear in the judgements. Another, more interesting, difference is the fact that by imposing the side-condition $x\ \#\ (s,\ t)$ in the rule rule QAT-Arrow we explicitly restricting the algorithm to consider only fresh variables. Recall that we imposed a similar restriction in the rule Q-Beta given in Sec. 3. There, however, the side-condition was innocuous as we could show that the rule *with* the side-condition is equivalent to the one *without* the side-condition. With rule QAT-Arrow the situation is different—the side-condition is a "real" restriction, meaning that

$$\frac{x\ \#\ \Gamma \qquad (x,\ T_1)::\Gamma \vdash App\ s\ (Var\ x) \Leftrightarrow App\ t\ (Var\ x) : T_2}{\Gamma \vdash s \Leftrightarrow t : T_1 \rightarrow T_2}$$

and QAT-Arrow are *not* interderivable. (The reason for this is that in the judgement $\Gamma \vdash s \Leftrightarrow t : T$ the free variables of s and t do not necessarily need to be contained in Γ. Therefore we cannot infer from $x\ \#\ \Gamma$ that $x\ \#\ (s,\ t)$ holds, as we did with Q-Beta.) While this restriction seems reasonable from an algorithmic point of view, it will turn out that it is actually crucial in our proofs: in order to get the inductions through for the properties of transitivity and monotonicity for the rules given above, we like to assume a sort of variable convention for x. That means we like to structure our argument so that the x in case of QAT-Arrow is fresh not just for Γ, s and t, but also for some other variables specified in the lemma at hand. This is very much like the informal reasoning using the variable convention, except that x in QAT-Arrow is not a binder. Still the nominal datatype package is able to derive automatically a strong induction principle, which allows us later on to make the reasoning with the variable convention completely formal. One proviso for deriving this strong induction principle is however that we formulate the QAT-Arrow as we have (essentially we have to make sure that the variable x does not occur freely in the conclusion of the corresponding rule; for more details we refer again to [10]). To see the improvement we obtain with the strong induction principle, consider the

usual induction principle that comes with the rules specified above:

$$\frac{\begin{array}{l}\ldots\\ \forall x\,\Gamma\,s\,t\,T_1\,T_2.\\ \quad x\#(\Gamma,s,t)\ \wedge\ P_1\,((x:T_1)::\Gamma)\,(App\,s\,(Var\,x))\,(App\,t\,(Var\,x))\,T_2\\ \qquad\qquad\qquad\qquad\qquad\qquad\longrightarrow P_1\,\Gamma\,s\,t\,(T_1\to T_2)\\ \ldots\end{array}}{\Gamma\vdash s\Leftrightarrow t:T\longrightarrow P_1\,\Gamma\,s\,t\,T\qquad \Gamma\vdash s\leftrightarrow t:T\longrightarrow P_2\,\Gamma\,s\,t\,T}$$

This induction principle states that if one wants to prove two properties P_1 and P_2 by mutual induction over the rules for algorithmic term equivalence and algorithmic path equivalence, then one can assume in the QAT-Arrow the side-condition $x\#(\Gamma,s,t)$ and P_1 for the premise, and one has to establish P_1 for the conclusion. The strong induction principle is similar

$$\frac{\begin{array}{l}\ldots\\ \forall x\,\Gamma\,s\,t\,T_1\,T_2\,c.\\ \quad x\#c\ \wedge\ x\#(\Gamma,s,t)\ \wedge\ (\forall c.\ P_1\,c\,((x:T_1)::\Gamma)\,(App\,s\,(Var\,x))\,(App\,t\,(Var\,x))\,T_2)\\ \qquad\qquad\qquad\qquad\qquad\qquad\longrightarrow P_1\,c\,\Gamma\,s\,t\,(T_1\to T_2)\\ \ldots\end{array}}{\Gamma\vdash s\Leftrightarrow t:T\longrightarrow P_1\,c\,\Gamma\,s\,t\,T\qquad \Gamma\vdash s\leftrightarrow t:T\longrightarrow P_2\,c\,\Gamma\,s\,t\,T}$$

except that it includes an induction context c in the properties P_1 and P_2, and we can assume that in the QAT-Arrow-case the x is fresh with respect to this induction context (see highlighted box). Over this induction context we have control when we set up an induction: if we want to employ the variable convention in our formal proofs, we just need to instantiate this induction context appropriately.

Before we can describe our proofs in detail we need two more definitions. We need to formalise Crary's notion of logical equivalence, written $\Gamma\vdash s\ is\ t:T$, and the logical equivalence of two simultaneous substitutions, say θ_1 and θ_2, over a context Γ. The latter is a derived concept and will be written as $\Gamma'\vdash\theta\ is\ \theta'\ over\ \Gamma$. The former is defined by recursion over the size of the types. The clauses are as follows:

$$\Gamma\vdash s\ is\ t:TUnit\stackrel{\text{def}}{=}true$$

$$\Gamma\vdash s\ is\ t:TBase\stackrel{\text{def}}{=}\Gamma\vdash s\Leftrightarrow t:TBase$$

$$\Gamma\vdash s\ is\ t:(T_1\to T_2)\stackrel{\text{def}}{=}\forall\Gamma'\,s'\,t'.\ \Gamma\subseteq\Gamma'\wedge valid\ \Gamma'\wedge\Gamma'\vdash s'\ is\ t':T_1\longrightarrow \Gamma'\vdash(App\ s\ s')\ is\ (App\ t\ t'):T_2$$

using in the last clause the notion of a weaker context, written $\Gamma\subseteq\Gamma'$ (for Γ' to be weaker than Γ, every (name,type)-pair in Γ must also appear in Γ'). Logical equivalence for simultaneous substitutions over a context Γ is defined as

$$\Gamma'\vdash\theta\ is\ \theta'\ over\ \Gamma\stackrel{\text{def}}{=}\forall x\ T.\ (x,T)\in set\ \Gamma\longrightarrow\Gamma'\vdash\theta(Var\,x)\ is\ \theta'(Var\,x):T$$

With this we have all necessary definitions in place.

5 Proofs

The first mayor property we need to establish is transitivity for algorithmic term equivalence and algorithmic path equivalence. These proofs are not detailed in Crary's notes and we diverged in our formalisation from the proofs he had in mind. We first show that type unicity holds for algorithmic path equivalence

Lemma 5.1 (Type Unicity)
If $\Gamma \vdash s \leftrightarrow t : T$ and $\Gamma \vdash s \leftrightarrow u : T'$ then $T = T'$.

and subsequently show symmetry for both the algorithmic path equivalence and the algorithmic term equivalence.

Lemma 5.2 (Algorithmic Symmetry)
If $\Gamma \vdash s \Leftrightarrow t : T$ then $\Gamma \vdash t \Leftrightarrow s : T$.
If $\Gamma \vdash s \leftrightarrow t : T$ then $\Gamma \vdash t \leftrightarrow s : T$.

Both proofs are by relatively straightforward inductions over $\Gamma \vdash s \Leftrightarrow t : T$ and $\Gamma \vdash s \leftrightarrow t : T$. This then allows us to prove transitivity, where we need the strong induction principle in order to get the induction through.

Lemma 5.3 (Algorithmic Transitivity)
If $\Gamma \vdash s \Leftrightarrow t : T$ and $\Gamma \vdash t \Leftrightarrow u : T$ then $\Gamma \vdash s \Leftrightarrow u : T$.
If $\Gamma \vdash s \leftrightarrow t : T$ and $\Gamma \vdash t \leftrightarrow u : T$ then $\Gamma \vdash s \leftrightarrow u : T$.

Proof. By mutual induction over $\Gamma \vdash s \Leftrightarrow t : T$ and $\Gamma \vdash s \leftrightarrow t : T$ where we instantiate the induction context with the term u. In the QAP-App-case we then have the induction hypotheses

$$ih_1: \quad \forall u.\ \Gamma \vdash q \leftrightarrow u : T_1 \to T_2 \longrightarrow \Gamma \vdash p \leftrightarrow u : T_1 \to T_2$$

$$ih_2: \quad \forall u.\ \Gamma \vdash t \Leftrightarrow u : T_1 \longrightarrow \Gamma \vdash s \Leftrightarrow u : T_1$$

and the assumptions

$$(i):\ \Gamma \vdash App\ q\ t \leftrightarrow u : T_2 \quad \text{and} \quad (ii):\ \Gamma \vdash p \leftrightarrow q : T_1 \to T_2$$

From the first assumption we obtain by inversion of the typing rule an r, T'_1 and v such that

$$(iii):\ \Gamma \vdash q \leftrightarrow r : T'_1 \to T_2 \quad (iv):\ \Gamma \vdash t \Leftrightarrow v : T'_1$$

and $u = App\ r\ v$ hold. From the second assumption we obtain $\Gamma \vdash q \leftrightarrow p : T_1 \to T_2$ by symmetry of $\leftrightarrow$ (Lemma 5.2), and then can use this and (iii) to find out by the type unicity of $\leftrightarrow$ (Lemma 5.1) that $T'_1 \to T_2 = T_1 \to T_2$ holds. This in turn implies that $T'_1 = T_1$, which allows us to use (iii) and (iv) in the induction hypotheses. This gives us

$$\Gamma \vdash s \Leftrightarrow v : T_1 \quad \text{and} \quad \Gamma \vdash p \leftrightarrow r : T_1 \to T_2\ .$$

Hence we know that $\Gamma \vdash App\ p\ s \leftrightarrow u : T_2$ holds by the rule QAP-App and the equation $u = App\ r\ v$.

The case QAT-Base uses the fact that normalisation produces unique results, that is if $t \Downarrow q$ and $t \Downarrow q'$ then $q = q'$.

In the QAT-Arrow case we have $\Gamma \vdash t \Leftrightarrow u : T_1 \rightarrow T_2$ and thus can infer that the judgement $(x,\ T_1)::\Gamma \vdash App\ t\ (Var\ x) \Leftrightarrow App\ u\ (Var\ x) : T_2$ holds. By induction we obtain further that $(x,\ T_1)::\Gamma \vdash App\ s\ (Var\ x) \Leftrightarrow App\ u\ (Var\ x) : T_2$ holds. Finally we can infer the proof obligation in this case, namely $\Gamma \vdash s \Leftrightarrow u : T_1 \rightarrow T_2$, provided we know $x \# (\Gamma,s,u)$. The freshness for Γ and s is given by the side-conditions of QAT-Arrow. The freshness for u is given by the strong induction principle (recall that we instantiated the induction context with u). Thus we are done. □

Next we prove closure under weak-head reductions, but we restrict our argument to the single step case (Crary proves closure under multiple steps) as this is easier to prove (actually it can be derived automatically by Isabelle's automatic search tools) and is sufficient for our formalisation.

Lemma 5.4 (Algorithmic Weak-Head Closure)
If $\Gamma \vdash s \Leftrightarrow t : T$ and $s' \rightsquigarrow s$ and $t' \rightsquigarrow t$ then $\Gamma \vdash s' \Leftrightarrow t' : T$.

This lemma is by a simple induction over $\Gamma \vdash s \Leftrightarrow t : T$. The following lemma establishes a kind of weakening property for the judgements of the algorithm.

Lemma 5.5 (Algorithmic Monotonicity)
If $\Gamma \vdash s \Leftrightarrow t : T$ and $\Gamma \subseteq \Gamma'$ and $valid\ \Gamma'$ then $\Gamma' \vdash s \Leftrightarrow t : T$.
If $\Gamma \vdash s \leftrightarrow t : T$ and $\Gamma \subseteq \Gamma'$ and $valid\ \Gamma'$ then $\Gamma' \vdash s \leftrightarrow t : T$.

Proof. By mutual induction using the strong induction principle. This time we instantiate the induction context with Γ'. The only interesting case (that is the one which is not automatic) analyses the rule QAT-Arrow: There we have by assumption $\Gamma \subseteq \Gamma'$ from which we can infer $(x,T_1)::\Gamma \subseteq (x,T_1)::\Gamma'$. In order to apply the induction hypotheses, we need the fact that $valid\ ((x,\ T_1)::\Gamma')$ holds. At this point the usual induction would start to become ugly since explicit renamings need to be performed. However we based our argument on the strong induction principle with the induction context being instantiated with Γ'. This gives us $x\ \#\ \Gamma'$ from which we can easily infer the desired fact. We can then conclude in this case with appealing to the induction hypotheses. □

The next lemma will help us to establish the fact that logical equivalence implies algorithmic equivalence.

Lemma 5.6 (Algorithmic Path Equivalence implies Weak-Head-Normal Form)
If $\Gamma \vdash s \leftrightarrow t : T$ then $s \not\rightsquigarrow$ and $t \not\rightsquigarrow$.

This is by straightforward induction on $\Gamma \vdash s \leftrightarrow t : T$. The main lemma in Crary's proof is then stated as follows (where we had to include in our formal version of this lemma that Γ is valid).

Lemma 5.7 (Main Lemma)
If $\Gamma \vdash s \; is \; t : T$ and $valid \; \Gamma$ then $\Gamma \vdash s \Leftrightarrow t : T$.
If $\Gamma \vdash p \leftrightarrow q : T$ then $\Gamma \vdash p \; is \; q : T$.

Proof. The proof is by simultaneous induction over T generalising over Γ, s, t, p and q. The non-trivial case is for $T = T_1 \rightarrow T_2$. For the first property we have the induction hypotheses

$$ih_1: \quad \forall \Gamma \; s \; t. \; \Gamma \vdash s \; is \; t : T_2 \wedge valid \; \Gamma \longrightarrow \Gamma \vdash s \Leftrightarrow t : T_2$$

$$ih_2: \quad \forall \Gamma \; s \; t. \; \Gamma \vdash s \leftrightarrow t : T_1 \longrightarrow \Gamma \vdash s \; is \; t : T_1$$

and the assumptions $\Gamma \vdash s \; is \; t : T_1 \rightarrow T_2$ and $valid \; \Gamma$. We choose a fresh x (fresh w.r.t. Γ, s and t). We can thus derive that $valid \; ((x, \; T_1)::\Gamma)$ holds and hence $(x, \; T_1)::\Gamma \vdash Var \; x \leftrightarrow Var \; x : T_1$. From this we can derive $(x, \; T_1)::\Gamma \vdash Var \; x \; is \; Var \; x : T_1$ using the second induction hypothesis. Using the our assumptions we can then derive $(x, \; T_1)::\Gamma \vdash App \; s \; (Var \; x) \; is \; App \; t \; (Var \; x) : T_2$ which by the first induction hypothesis leads to $(x, \; T_1)::\Gamma \vdash App \; s \; (Var \; x) \Leftrightarrow App \; t \; (Var \; x) : T_2$. Because we chosen x to be fresh, we can then derive $\Gamma \vdash s \Leftrightarrow t : T_1 \rightarrow T_2$, as needed. The proof for the other property uses Lemma 5.5, but we omit the details. □

In his notes Crary carefully designs the logical equivalence so that it has the following property:

Lemma 5.8 (Logical Monotonicity)
If $\Gamma \vdash s \; is \; t : T$ and $\Gamma \subseteq \Gamma'$ and $valid \; \Gamma'$ then $\Gamma' \vdash s \; is \; t : T$.

whose proof is by induction on the definition of the logical equivalence, appealing in the *TBase*-case to Lemma 5.5. From logical monotonicity we can deduce the following corollary:

Corollary 5.9 (Logical Monotonicity for Substitutions)
If $\Gamma' \vdash \theta \; is \; \theta' : \Gamma$ and $\Gamma' \subseteq \Gamma''$ and $valid \; \Gamma''$ then $\Gamma'' \vdash \theta \; is \; \theta' : \Gamma$.

The next three lemmas infer some properties about logical equivalence needed in the fundamental theorems. They are all by relatively routine inductions over the type T, so we only state them here.

Lemma 5.10 (Logical Symmetry)
If $\Gamma \vdash s \; is \; t : T$ then $\Gamma \vdash t \; is \; s : T$.

Lemma 5.11 (Logical Transitivity)
If $\Gamma \vdash s \; is \; t : T$ and $\Gamma \vdash t \; is \; u : T$ then $\Gamma \vdash s \; is \; u : T$.

Lemma 5.12 (Logical Weak Head Closure)
If $\Gamma \vdash s \; is \; t : T$ and $s' \rightsquigarrow s$ and $t' \rightsquigarrow t$ then $\Gamma \vdash s' \; is \; t' : T$.

Note that in Lemma 5.12 we prove again only the case of closure under single weak-head reductions since this is sufficient for the the fundamental theorems, which are shown next.

Theorem 5.13 (Fundamental Theorem 1)
If $\Gamma \vdash t : T$ and $\Gamma' \vdash \theta \; is \; \theta' : \Gamma$ and $valid \; \Gamma'$ then $\Gamma' \vdash \theta(t) \; is \; \theta'(t) : T$.

Proof. By induction over the definition of $\Gamma \vdash t : T$. We use the strong induction principle for typing and instantiate the induction context so that binders avoid the substitutions θ and θ'. This will give us the two facts

$$(*) \quad (x, s)::\theta(t) = \theta(t)[x:=s] \qquad \text{and} \qquad (x, s)::\theta'(t) = \theta'(t)[x:=s]$$

which state how we can pull apart a simultaneous substitution such that we obtain a separate single substitution. These facts will be crucial in our induction argument since the left-hand sides correspond to what we have by the induction hypotheses and the right-hand sides will correspond to what we have to prove. These facts do, however, not hold for general x, only for ones that are fresh for the substitution. Since we can assume that x is fresh for θ and θ', our argument goes through smoothly. In the T-Lam-case we have the induction hypothesis

$$ih: \quad \forall \Gamma' \; \theta \; \theta'. \; \Gamma' \vdash \theta \; is \; \theta' : (x, T_1)::\Gamma \wedge valid \; \Gamma' \longrightarrow \Gamma' \vdash \theta(t_2) \; is \; \theta'(t_2) : T_2$$

and we can assume $\Gamma' \vdash \theta \; is \; \theta' : \Gamma$ and further that $x \; \# \; (\Gamma, \theta, \theta')$ (the first freshness assumption comes from the T-Lam rule; the second and third from the strong induction). We need to show that $\Gamma' \vdash \theta(Lam \; x.t_2) \; is \; \theta'(Lam \; x.t_2) : T_1 \rightarrow T_2$ holds. For this it is sufficient to show for all Γ'', s' and t' that

$$\Gamma'' \vdash App \; (Lam \; x.\theta(t_2)) \; s' \; is \; App \; (Lam \; x.\theta'(t_2)) \; t' : T_2$$

whereby we can assume that $\Gamma' \subseteq \Gamma''$, $\Gamma'' \vdash s' \; is \; t' : T_1$ and $valid \; \Gamma''$. From these assumptions we obtain by Lemma 5.8 that $\Gamma'' \vdash \theta \; is \; \theta' : \Gamma$ holds and by the freshness conditions also that $\Gamma'' \vdash (x, s')::\theta \; is \; (x, t')::\theta' : (x, T_1)::\Gamma$ (we proved that logical equivalence can be so extended in a side-lemma). Now by induction hypothesis we can infer that

$$\Gamma'' \vdash (x, s')::\theta(t_2) \; is \; (x, t')::\theta'(t_2) : T_2$$

holds. Now we can apply the facts mentioned under $(*)$ to obtain

$$\Gamma'' \vdash \theta(t_2)[x:=s'] \; is \; \theta'(t_2)[x:=t'] : T_2$$

Since we know that

$$App \; (Lam \; x.\theta(t_2)) \; s' \rightsquigarrow \theta(t_2)[x:=s']$$

$$App \; (Lam \; x.\theta'(t_2)) \; t' \rightsquigarrow \theta'(t_2)[x:=t']$$

hold, we can apply Lemma 5.12 to conclude with $\Gamma'' \vdash App \; (Lam \; x.\theta(t_2)) \; s' \; is \; App \; (Lam \; x.\theta'(t_2)) \; t' : T_2$. This completes, the proof as the T-Lam-case is the only non-automatic case in our formal proof. □

The second fundamental lemma shows that logical equivalence is closed under simultaneous substitutions.

Theorem 5.14 (Fundamental Theorem 2)
If $\Gamma \vdash s \equiv t : T$ and $\Gamma' \vdash \theta \; is \; \theta' : \Gamma$ and $valid \; \Gamma'$ then $\Gamma' \vdash \theta(s) \; is \; \theta'(t) : T$.

Proof. By strong induction over the definition of the definitional equivalence $\Gamma \vdash s \equiv t : T$. The induction context is again instantiated with θ and θ'. There are several interesting cases. However we only show the cases for Q-Abs, Q-Beta and Q-Ext.

In the first case we have the induction hypothesis

$$ih: \quad \forall \Gamma' \; \theta \; \theta'. \; \Gamma' \vdash \theta \; is \; \theta' : (x, \; T_1)::\Gamma \wedge valid \; \Gamma' \longrightarrow \Gamma' \vdash \theta(s_2) \; is \; \theta'(t_2) : T_2$$

and need to show that

$$\Gamma' \vdash \theta(Lam \; x.s_2) \; is \; \theta'(Lam \; x.t_2) : T_1 \rightarrow T_2$$

holds. Because by the strong induction principle, we can assume that $x \; \# \; (\theta, \theta')$, we are able to immediately move the substitutions under the lambdas, i.e. we have to proceed with showing

$$\Gamma' \vdash Lam \; x.\theta(s_2) \; is \; Lam \; x.\theta'(t_2) : T_1 \rightarrow T_2.$$

This can be done by establishing $\Gamma'' \vdash App \; (Lam \; x.\theta(s_2)) \; s' \; is \; App \; (Lam \; x.\theta'(t_2)) \; t' : T_2$ for all Γ'', s' and t'. The reasoning is similar to Theorem 5.13 and therefore omitted.

In the second case we need to establish that $\Gamma' \vdash \theta(App \; (Lam \; x.s_1) \; s_2) \; is \; \theta'(t_1[x::=t_2]) : T_2$ holds. Again, by the convenience afforded by the strong induction principle we can immediately move the substitution inside the terms, that is we have to show

$$\Gamma' \vdash App \; (Lam \; x.\theta(s_1)) \; \theta(s_2) \; is \; \theta'(t_1)[x::=\theta'(t_2)] : T_2$$

We omit the other details, because they just amount to using the induction hypotheses and adjusting substitutions appropriately.

In the third case we do not have additional freshness assumptions about θ and θ' (we do not need them in this case). However, the side-conditions about x being fresh for s and t will turn out to be crucial. The reason is that we can then simplify the terms

$$(**) \quad (x, \; s')::\theta(s) = \theta(s) \qquad \text{and} \qquad (x, \; t')::\theta'(t) = \theta'(t)$$

The induction hypothesis in this case is

$$\forall \Gamma' \; \theta \; \theta'. \; \Gamma' \vdash \theta \; is \; \theta' \; over \; (x,T_1)::\Gamma \wedge valid \; \Gamma' \longrightarrow \Gamma' \vdash \theta(App \; s \; (Var \; x)) \; is \; \theta'(App \; t \; (Var \; x)) : T_2.$$

and we have the assumptions that $\Gamma' \vdash \theta \; is \; \theta' : \Gamma$, $valid \; \Gamma'$ and $x \; \# \; (\Gamma, \; s, \; t)$. We show that $\Gamma' \vdash \theta(s) \; is \; \theta'(t) : T_1 \rightarrow T_2$ holds which by the assumption about the validity of Γ' amounts to showing that

$$\Gamma'' \vdash App\ \theta(s)\ s'\ is\ App\ \theta'(t)\ t' : T_2$$

holds for all Γ'', s' and t', using the assumption about $\Gamma' \subseteq \Gamma''$, $\Gamma'' \vdash s'\ is\ t' : T_1$ and *valid* Γ''. Using Lemma 5.8 we can infer that

$$\Gamma'' \vdash \theta\ is\ \theta' : \Gamma$$

holds, from which we obtain

$$\Gamma'' \vdash (x,\ s')::\theta\ is\ (x,\ t')::\theta' : (x,\ T_1)::\Gamma.$$

Using the induction hypothesis gives us then

$$\Gamma'' \vdash (x,\ s')::\theta(App\ s\ (Var\ x))\ is\ (x,\ t')::\theta'(App\ t\ (Var\ x)) : T_2.$$

Moving the substitutions inside and using the facts $(**)$ we can conclude with

$$\Gamma'' \vdash App\ \theta(s)\ s'\ is\ App\ \theta'(t)\ t' : T_2$$

This completes the proof. □

Completeness of the algorithm is now a simple consequence of the Theorem 5.14 by using the fact that $\Gamma \vdash Var\ x\ is\ Var\ x : T$ holds by Lemma 5.7 and that $\Gamma \vdash []\ is\ [] : \Gamma$ holds.

Corollary 5.15 (Completeness)
If $\Gamma \vdash s \equiv t : T$ then $\Gamma \vdash s \Leftrightarrow t : T$.

Thus we have formally verified that the algorithm says "yes" for all equivalent terms. The soundness property is left as an exercise in [3]. We have not formalised this part.

6 About the Formalisation

We can generally remark that having a formalised proof allows one to quickly test changes whether they affect the whole proof. This proved convenient for testing if lemmas or definitions need to be strengthened or can be weakened. Having the formal proof at our disposal also made it easy to compile this paper, as Isabelle has an extensive infrastructure for using formal definitions in papers and providing sanity checks. This is especially useful to keep formalisations and papers synchronised. The inductive rules and the statements of the lemmas and theorems presented in this paper have been generated from the formal definitions.

More specifically we can say that our formalisation follows a good deal the informal reasoning of Crary (see Figure 1 which shows the first fundamental lemma as an example in the Isar proof language [12]). The strong induction principles proved crucial in order to get the inductions through. Such strong induction principles are derived automatically for any nominal datatype (which can at the moment only include lambda-type of bindings, but they can occur iterated and can bind different kinds of variables). The strong induction principles are also derived automatically

for inductive definition satisfying certain freshness conditions (see [10]).

The only sore point we see in our formalisation is the lack of automation in inversion lemmas. While this is not a serious problem in the formalisation of Crary's chapter (we only need one such inversion lemma and that can be proved in 5 lines), it can be painful in other formalisations. We hope this problem can be solved in the future. To see what the issues are, re-consider the T-Lam-rule:

$$\frac{x \ \# \ \Gamma \qquad (x,\ T_1)::\Gamma \vdash t : T_2}{\Gamma \vdash \mathit{Lam}\ x.t : T_1 \rightarrow T_2}\text{T-Lam}$$

and assume that we have given the typing judgement $\Gamma \vdash \mathit{Lam}\ x.t : T$. In in formal reasoning we can match this judgement against all typing rules, which is only successful in case of T-Lam. The informal matching would then produce that there exists an T_1 and T_2 such that $T = T_1 \rightarrow T_2$ and that $(x,\ T_1)::\Gamma \vdash t : T_2$ as well as $x \ \# \ \Gamma$ hold. However, this is not how we can proceed in the nominal datatype package, where terms are α-equivalent classes. There we obtain for the assumption $\Gamma \vdash \mathit{Lam}\ x.t : T$ the "matcher" that there exists Γ', x', t', T_1' and T_2' so that $\Gamma = \Gamma'$, $\mathit{Lam}\ x.t = \mathit{Lam}\ x'.t'$ and $T = T_1' \rightarrow T_2'$. As properties we obtain $\Gamma' \vdash \mathit{Lam}\ x'.t : T'$ and $x' \ \# \ \Gamma'$. Solving these equation would be no problem if we had term-constructors that are injective (that is a characteristics of standard, unquotioned datatypes). However, our constructors for α-equivalence classes are clearly *not* injective. What we have to do is to analyse $\mathit{Lam}\ x.t = \mathit{Lam}\ x'.t'$ according to the built-in notion of the nominal datatype package for α-equivalence. We end up with two cases: one is simple and the other needs explicit renamings. However these reasoning maneuvers should really be performed automatically by the nominal datatype package.

7 Conclusion

We presented a formalisation of Crary's case study about logical relations. This is in addition to the usual strong normalisation proof of the simply-typed lambda-calculus, which has been part of the nominal datatype package for quite some time. It remains to be seen whether the nominal datatype package is up to the task of formalising strong normalisation for System F, where also types have binders. In this case the definition of logical relations is not completely trivial like in the completeness proof we presented above.

We are aware of work by Schürmann and Sarnat about formalising logical relation proofs in Twelf [7]. This involves a clever trick of implementing an object logic in Twelf and coding the logical relation proof in this object logic. We unfortunately do not know how convenient this style of reasoning is. We are also aware that Aydemir et al [2] use a locally nameless approach (which goes back to work by McKinna and Pollack [5]) to representing binders and work on formalising programming language theory. It would be interesting to compare in detail our formalisation and the approach taken by Aydemir et al. Our initial opinion is that in our formalisation we do not have to deal with the concepts of *open* and *closed*

theorem *fundamental-theorem-1*:
  assumes a_1: $\Gamma \vdash t : T$
  and a_2: $\Gamma' \vdash \theta$ *is* θ' *over* Γ
  and a_3: *valid* Γ'
  shows $\Gamma' \vdash \theta(t)$ *is* $\theta'(t) : T$
using a_1 a_2 a_3
proof (*nominal-induct* Γ t T *avoiding*: θ θ' *arbitrary*: Γ' *rule*: *typing.strong-induct*) *(**)*
  case (*T-Lam* x Γ T_1 t_2 T_2 θ θ' Γ')
  have *vc*: $x \# \theta$ $x \# \theta'$ **by** *fact* *(variable convention)*
  have *fs*: $x \# \Gamma$ **by** *fact* *(freshness condition from the rule)*
  have *asm*$_1$: $\Gamma' \vdash \theta$ *is* θ' *over* Γ **by** *fact*
  have *ih*:$\bigwedge \theta\ \theta'\ \Gamma'.\ [\![\Gamma' \vdash \theta$ *is* θ' *over* $(x,T_1)::\Gamma$; *valid* $\Gamma']\!] \Longrightarrow \Gamma' \vdash \theta(t_2)$ *is* $\theta'(t_2) : T_2$
    by *fact* *(induction hypothesis)*
  show $\Gamma' \vdash \theta(Lam\ x\ .\ t_2)$ *is* $\theta'(Lam\ x\ .\ t_2) : T_1 \rightarrow T_2$ **using** *vc* *(*)*
  proof (*simp*, *intro strip*) *(unfolding the definition of logical equivalence)*
    fix Γ'' s' t'
    assume *sub*: $\Gamma' \subseteq \Gamma''$
    and *asm*$_2$: $\Gamma'' \vdash s'$ *is* $t' : T_1$
    and *val*: *valid* Γ''
    from *asm*$_1$ *val sub* **have** $\Gamma'' \vdash \theta$ *is* θ' *over* Γ **using** *logical-subst-monotonicity* **by** *blast*
    with *asm*$_2$ *vc fs* **have** $\Gamma'' \vdash (x,s')::\theta$ *is* $(x,t')::\theta'$ *over* $(x,T_1)::\Gamma$ *(*)*
      using *equiv-subst-ext* **by** *blast*
    with *ih val* **have** $\Gamma'' \vdash ((x,s')::\theta)(t_2)$ *is* $((x,t')::\theta')(t_2) : T_2$ **by** *auto*
  with *vc* **have** $\Gamma'' \vdash \theta(t_2)[x::=s']$ *is* $\theta'(t_2)[x::=t'] : T_2$ **by** (*simp add*: *psubst-subst*) *(*)*
    moreover
    have *App* (*Lam* x . $\theta(t_2)$) $s' \leadsto \theta(t_2)[x::=s']$ **by** *auto*
    moreover
    have *App* (*Lam* x . $\theta'(t_2)$) $t' \leadsto \theta'(t_2)[x::=t']$ **by** *auto*
    ultimately show $\Gamma'' \vdash$ *App* (*Lam* x . $\theta(t_2)$) s' *is App* (*Lam* x . $\theta'(t_2)$) $t' : T_2$
      using *logical-weak-head-closure* **by** *auto*
  qed
qed (*auto*) *(all other cases are automatic)*

Fig. 1. The complete formalised proof of the first fundamental lemma (Lemma 5.13) in the readable Isar proof-language. In the places marked with a single star, one appeals in informal reasoning to the variable convention about the binder x. This variable convention is given in our proof by the strong induction principle and by declaring that x should avoid θ and θ' (see line marked with two stars). The fact *logical-subst-monotonicity* is Corollary 5.9; *equiv-subst-ext* establishes that for a fresh x one can extend the logical equivalence of simultaneous substitutions; and *psubst-subst* is a lemma that allows us to pull apart a simultaneous substitution in order to obtain a single substitution. We can do this provided the variable convention about x holds.

terms; and that we do not have to discard any *pre-terms*.

The sources of our formalisation are included in the nominal datatype package (see http://isabelle.in.tum.de/nominal/). From the web-page of the first author one can also download a longer version of the documented proofs.

Acknowledgement

We thank Karl Crary for the discussions about his proof. We are also very grateful to Carsten Schürmann who made us aware of typos and omissions in an early version of this paper.

References

[1] T. Altenkirch. A Formalization of the Strong Normalisation Proof for System F in LEGO. In *Proc. of TLCA*, volume 664 of *LNCS*, pages 13–28, 1993.

[2] B. Aydemir, A. Charguéraud, B. C. Pierce, and S. Weirich. Engineering Aspects of Formal Metatheory, 2007. Submitted for publication.

[3] K. Crary. Logical Relations and a Case Study in Equivalence Checking. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, pages 223–244. MIT Press, 2005.

[4] R. Harper and D. Licata. Mechanizing Metatheory in a Logical Framework. *Journal of Functional Programming*, 2007. To appear.

[5] J. McKinna and R. Pollack. Pure Type Systems Formalized. In *Proc. of the International Conference on Typed Lambda Calculi and Applications (TLCA)*, number 664 in LNCS, pages 289–305. Springer-Verlag, 1993.

[6] A. M. Pitts. Nominal Logic, A First Order Theory of Names and Binding. *Information and Computation*, 186:165–193, 2003.

[7] C. Schürmann and J. Sarnat. Towards a Judgemental Reconstruction of Logical Relation Proofs. Submitted, 2007.

[8] W. W. Tait. Intensional Interpretations of Functionals of Finite Type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

[9] C. Urban and S. Berghofer. A Recursion Combinator for Nominal Datatypes Implemented in Isabelle/HOL. In *Proc. of the 3rd International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *LNAI*, pages 498–512, 2006.

[10] C. Urban, S. Berghofer, and M. Norrish. Barendregt's Variable Convention in Rule Inductions. In *Proc. of the 21th International Conference on Automated Deduction (CADE)*, 2007. To appear.

[11] C. Urban and C. Tasson. Nominal Techniques in Isabelle/HOL. In *Proc. of the 20th International Conference on Automated Deduction (CADE)*, volume 3632 of *LNCS*, pages 38–53, 2005.

[12] M. Wenzel. Isar — A Generic Interpretative Approach to Readable Formal Proof Documents. In *Proc. of the 12th International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, number 1690 in LNCS, pages 167–184, 1999.